WHAT IS CLAIMED IS:

1. A method for measuring complexity comprising this steps of:

creating a graph representation of a set of use-cases comprising the steps of:

creating a plurality of vertices corresponding to start and/or end points of said use-cases; and

creating a plurality of edges corresponding to a plurality of said set of use-cases, each of said edges connecting two of said vertices; and

determining the complexity from the graph representation.

2. The method as in claim 1 wherein said plurality of use-cases comprise at least one normal use-case and at least one exception use-case.

3. The method as in claim 2 wherein at least one or more of said edges represents a portion of the at least one exception use-case that differs with the at least one normal use-case, with details of these edges represented in existing object/class interaction-scenario diagrams.

4. The method as in claim 3 wherein said creating one or more use-case set comprises the steps of:

creating the object/class scenario with the longest forward and backward processes; and

indicating their start/end node in said graph on claim 1.

5. The method as in claim 4 comprises the steps of creating one or more edges for each of the exception use-cases.

6. The method as in claim 5 wherein the same scenario segment can be reused.

7. The method as in claim 1, wherein said cyclomatic complexity is defined as

$$SBDLCM_{Mc} = SBDMc_0 + \sum_{i=1}^{m}\sum_{j=1}^{n_i}\sum_{k=1}^{l_{i,j}} Mc_{i,j,k}$$

wherein $Mc_{i,j,k}$ is the cyclomatic complexity of method k in the use-case j of use-case set i; $SBDMc_0$ is the sum of the cyclomatic complexity of all SBDs;

m = the number of use-case sets in the SBDs;

n_i = the number of use-cases in use-case set i; and

$l_{i,j}$ = the number of methods called in the use-case j of use-case set i.

8. A method as in claim 1 wherein said implementation-path complexity is defined as:

$$SBDLCM_{YPath} = \sum_{i=1}^{m}\sum_{j=1}^{n_i}\prod_{k=1}^{l_{i,j}} YPathM_{i,j,k}$$

wherein m=the number of use-case sets in the SBDs;

n_i=the number of use-cases in use-case set i;

$l_{i,j}$=the number of methods called in the use-case j of use-case set i; and

$YPathM_{i,j,k}$ is the number of execution paths in the method $M_{i,j,k}$ to keep the execution in the use-case j of use-case set i.

9. A method for measuring business-table complexity comprising the step of:

normalizing the business table according to normalization rules.

10. The method for business-table normalization as in claim 9, further comprising the step of:

dividing business-table cells into two types: the first type representing conditions (named if-cells) and second type to represent assignment (named then-cells); and

putting cells of the first type at left, cells of the second type at right.

11. The method as in 10 business-table normalization comprises the step of:

assigning each said second type cells s only one value.

12. The method as in claim 11 wherein said determining normalized business table comprises the step of:

arranging said cells of said first type in one or more columns in descending order of their coverage rate and importance.

13. The method as in claim 11 wherein said determining normalized business table step further comprises the step of :

merging adjacent ones of said one or more cells of said first type having identical values.

14. The method as in claim 10 further comprising the step of computing the complexity from said normalized business table.

15. A method as in claim 1wherein said complexity is defined as:

$$BTCMI_{MC} = BTCMI_{Path} = \sum_{k=1}^{m} p_k + \sum_{j=1}^{c} (1 + p_j) + 1$$

wherein m is the number of if-cells in one or more main columns of said at least one business table;

c is the number of if-cells of one or more comment cells;

p_k is the number of predicates in one of said if-cells k in said main columns; and

p_j is the number of predicates in one of said if-cellsj of said one or more comments cells

16. A method for measuring complexity of nested object state transition diagrams comprising the steps of:

determining a plurality of graphs of object state transitions at K levels l_k $0 < k < K$ wherein

one or more of said graphs at level l_{k+1} are expansions of one or more of said graphs at level l_k; and

said graphs comprise a plurality of nodes to represent a corresponding plurality of states of use-cases and a plurality of edges to represent a corresponding plurality of transitions between the states; and determining the complexity for said plurality of graphs.

17. The method as in claim 16 wherein the state of an obobject at level k is a super state of an object state at level k+1, and it isa sub state of an object state at level k-1 and includes the step of identifying the state transion relationship between super state and sub state as in one of the three cases as "in"/"out"/"inout".

18. The method as in claim 17 wherein said determining the number of paths comprises the steps of

selecting said level $l_k = l_K$;

expanding a graph L from said selected level l_k with said at least one graph from said level l_{K+1};

determining the paths in said expanded graph; and

determining the number of conditions in said transition paths.

19. The method as in claim 18 further comprising the step of removing one or more unnecessary paths from said at least one graph.

20. The method as in claim 19 wherein said one or more unnecessary paths comprise one or more numbers of the group consisting of (exception→ fallout), (exception → exception), (one-repetition loop →fallout), (one repetition loop → cancelled), and (one repetition loop → exception).

21. A method as in claim 16 to measure nested object state transition complexity between two super-states by recursively applying Equation 6:

$$STPC_{k,p,q} = \sum_{i=1}^{m_k} (\prod_{j=1}^{n_{k,i}} (C_{k,i,j} + (STPC_{k+1,i,j} - sub_{k,i,j})) + mul_{k,i}(N)) \qquad \textbf{Equation 6}$$

where m_k = the number of transition paths of level-k object (k>1) between two super-states or the number of transition paths of level-1 object;

$n_{k,i}$= the number of states along path i for level-k object.

$C_{k,i,j}$ is the number of conditions to bring level-k object state from $S_{k,i,j1}$ to $S_{k,i,j}$;

$STPC_{k+1,i,j}$ = Substate Transition Complexity between state $S_{k,i,j1}$ and $S_{k,i,j}$;

$sub_{k,i,j}$=0 if $STPC_{k+1,i,j}$=0 and $sub_{k,i,j}$=1 if ($STPC_{k+1,i,j}$≠0)

If the multiplicity between level-k objct and level-$_{(k+1)}$ is 1:N, then

$mul_{k,i}(N)$=0, if (N=1) and $mul_{k,i}(N)$=r, if (N>1);

Where r is the number of times that level-$_{k+1}$ objects transits back to level-$_k$ in path i.

22. The method for nested State Transition Diagram Logical Complexity Metric (STDLCM) through the state transition paths as:

$$STDLCM_{path} = STPC_{1,0,0} \qquad \textbf{Equation 7}$$

where 1 is the first level object.